Detecting Malicious Middleboxes In Service Function Chaining

Nguyen Canh Thang[1] and Minho Park[1,2*]
[1]Department of Information Communication Convergence Technology
Soongsil University, Seoul 156-743, South Korea
{nct, mhp}@soongsil.ac.kr
[2]School of Electronic Engineering, Soongsil University, Seoul 156-743, South Korea

Abstract

Service Function Chaining (SFC) has become a new and robust technology in computer networking, and takes advantage of both Software-Defined Networking (SDN) and Network Function Virtualization (NFV). However, SFC simultaneously inherited the vulnerabilities from SDN and NFV, especially the problem of malicious middleboxes. In this paper, we present a scheme that can detect malicious middleboxes in SFC by combining two mechanisms: direct and indirect. The direct mechanism injects a tool into the middleboxes to observe and report the state of each middlebox. In contrast, the indirect mechanism creates a probe service chain, which includes trustful middleboxes, to investigate the operation of other middleboxes in the network. Our experimental results show that the proposed system exhibits low resource consumption while achieving a high detection rate and accuracy. In addition, we demonstrate that the system is able to successfully detect malicious middleboxes in SFC.

Keywords: Service Function Chaining, Malicious Middlebox, Software-Defined Networking, Network Function Virtualization.

1 Introduction

Service Function Chaining (SFC), which has become a new and robust technology in computer networking, incorporates the advantages of both Software-Defined Networking (SDN) and Network Function Virtualization (NFV). This technique can create on-demand ordered chains of network services (e.g., the load balancer, firewalls, and network address translation), which it then uses to steer the network traffic to ensure that the network is agile and effective. Service functions run on middleboxes, which are connected to switches in the network, and SFC connects these switches to create the required virtual chains [1] –[12].

Because of the virtual attributes obtained from SDN and NFV, SFC is prone to encounter various security vulnerabilities, especially malicious middleboxes. In particular, attackers can modify the service functions that run on the middlebox or inject malicious code into the middlebox to perform harmful actions. Malicious middleboxes can create various attack types that exploit the weaknesses of both SDN and NFV to disrupt the operation and policy of SFC. With respect to the SDN, malicious middleboxes can attack the control and data plane by launching distributed denial-of-service (DDoS) attacks, abusing computing resources, or incorrectly managing the network traffic. With respect to the NFV, malicious middleboxes can attack the infrastructure of other middleboxes, or even user equipment or the network by injecting malware, spoofing or sniffing data, carrying out denial-of-service (DoS) attacks, misusing shared resources, violating the privacy and confidentiality, etc. [16] –[11].

Journal of Internet Services and Information Security (JISIS), volume: 10, number: 2 (May 2020), pp. 82-90
DOI: *10.22667/JISIS.2020.05.31.082*

*Corresponding author: Department of Information Communication Convergence Technology, Soongsil University, Seoul 156-743, South Korea, Tel: +82-(0)2-828-7176

Many countermeasures have been proposed to protect the network from these attacks, by either analyzing the network traffic [5] – [10] or by installing programs in virtual machines (VMs) to collect data generated by the hardware to discover the attacks [13] – [15]. However, in the SFC environment, these solutions still have limitations and vulnerabilities because they only focus on a specific type of attacks and their mechanisms can be detected and compromised by attackers. In this paper, we propose a scheme that can detect malicious middleboxes in SFC and is able to overcome the limitation of other methods. The proposed scheme functions by way of two mechanisms: direct and indirect, which make it possible to detect attacks launched both from the inside and outside of middleboxes. The direct mechanism injects a tool into the middleboxes to observe and report the state of each middlebox. This tool can discover abnormal action by detecting high resource consumption processes or determine whether an abnormal process is installed. On the other hand, the indirect mechanism creates a probe service chain, which includes trustful middleboxes, to investigate the operation of other middleboxes in the network.

In summary, the contributions of this paper are described as follows:

- We investigate the security vulnerabilities of SFC posed by malicious middleboxes.
- We propose a new combined scheme to detect malicious middleboxes in SFC.
- We implement and evaluate the efficiency of the mechanisms.

The remainder of this paper is organized as follows. Section 2 presents the work most closely related to our research. Details of our proposed scheme are provided in Section 3. Section 4 describes the implementation and evaluation of our scheme in detail. Finally, we conclude the paper and mention future research possibilities in Section 5.

2 Related Work

In this section, we present several proposed approaches related the most to our work. In terms of the direct method, Jamshed et al. [5] presented Kargus, a highly scalable software-based intrusion detection system IDS, which exploits the full potential of computing hardware to analyze the network traffic. Sherry [14] proposed BlindBox, a system that simultaneously performs deep-packet inspection by handling the encrypted packet payload. Modi [10] inserted an IDS (Snort) into every middlebox and used machine learning to analyze the network data. Shanmugam [13] presented distributed elastic intrusion detection architecture named DEIDtect and spread it across multiple platforms to detect the attacks. Ibrahim [4] installed a program named CloudSec in VMs to monitor the physical memory of the VMs. Watson [15]] introduced and discussed an online cloud anomaly detection approach, which extracts the raw features per process of VMs and uses machine learning to analyze the data. With respect to the indirect method, Kuo [6] proposed a detection mechanism and added ghost switches to the network to check the operation of the other entities. However, each of the aforementioned solutions only focused on a specific attack. For example, if only the network traffic is analyzed, only attacks related to packets would be detected (e.g., a packet incorrect forwarding attack or a manipulating attack). Alternatively, monitoring and analyzing only the features of a process would only detect attacks related to the process (e.g., resource abusing attacks and DoS). Furthermore, attackers could detect and compromise these mechanisms and adapt malicious middleboxes to attack them.

3 Our Proposal

3.1 Attack Models

In the network, a middlebox is typically created with a fresh operating system and service functions, except when the middlebox is created from a malicious image. To perform attacks, attackers either need to compromise the normal service functions or inject malicious programs into middleboxes, as illustrated in Fig. 1. The malicious middleboxes then acquire the ability to drop received packets, duplicate them to place additional burden on the next-hop middlebox (packet dropping, multiplying attack) or forward packets incorrectly to other middleboxes or even to attackers (eavesdropping, man-in-the-middle attacks). Furthermore, malicious middleboxes can run redundant processes, which abuse the resources of the middlebox and affect the operation of other service functions inside the middlebox. In this study, we only focused on solving the following attack situations: packet dropping, multiplying, eavesdropping, man-in-the-middle, and resource abusing attacks, which are depicted in Fig. 2. We assume that controllers, switches, and the connections between them are trustful. Attackers who succeed in gaining access to controllers or switches could exploit the network information and destroy all the detection and prevention mechanisms.

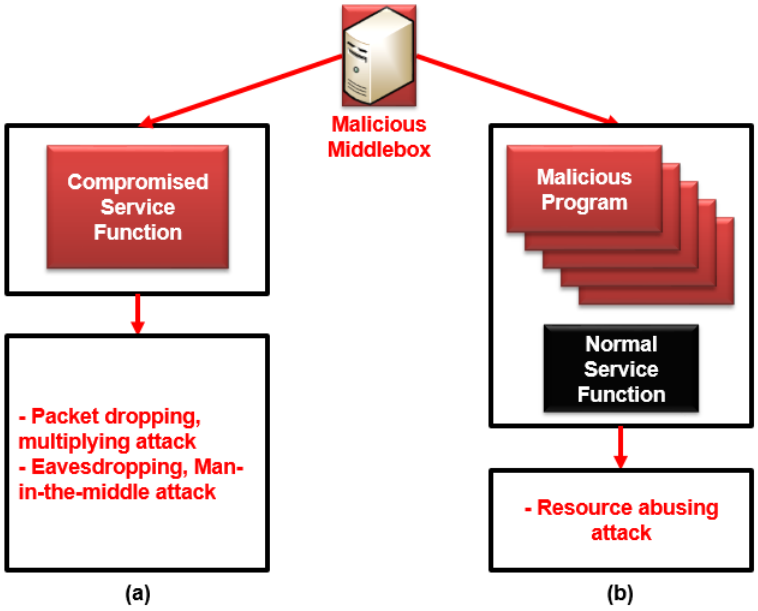


Figure 1: Attack models involving a malicious middlebox. Attackers a) compromise the normal service functions and b) inject malicious programs into middleboxes.

3.2 Methodology

Our proposed scheme creates and injects a malicious tracking tool into middleboxes to detect the attacks. By tracking the device information (the number of processes, % CPU and memory usage of each process, network traffic on a network interface, etc.), our tool can detect the above types of attacks. The malicious tracking tool model is illustrated in Fig. 3. The model contains the following three components:

- Resource Observation Module: tracks the number of processes and resources consumed by each process and sends the results to the Analyzing Module.

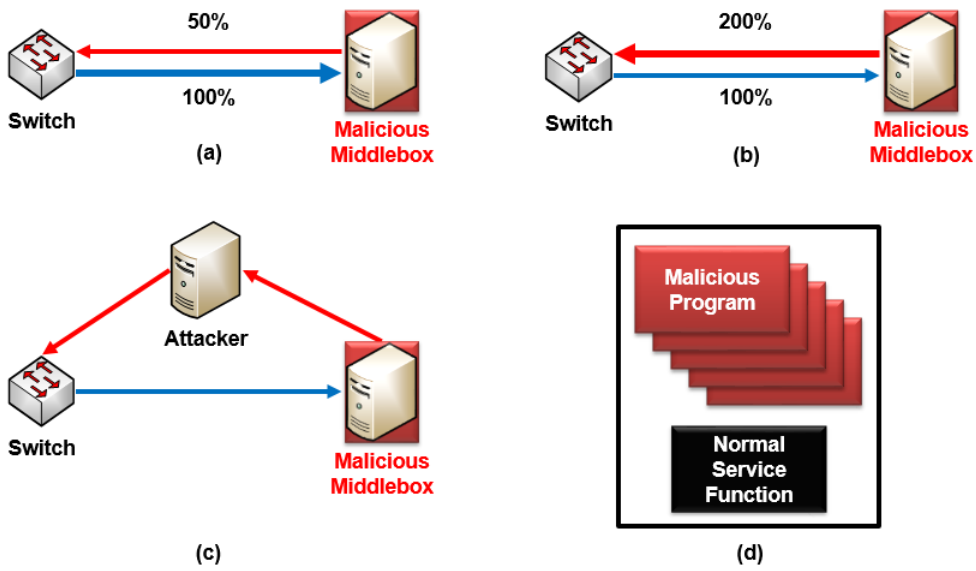


Figure 2: Examples of attack cases: a) Packet dropping attack, b) packet multiplying attack, c) eavesdropping, man-in-the-middle attack, d) resource abusing attack.

- Packet Observation Module: tracks the network traffic on network interfaces (the number of packets entering and exiting, transmission latency, etc.) and sends the results to the Analyzing Module.
- Analyzing Module: based on the tracking results from other modules, this module decides and raises alarms to the controller about any irregularities in the operation of middleboxes.

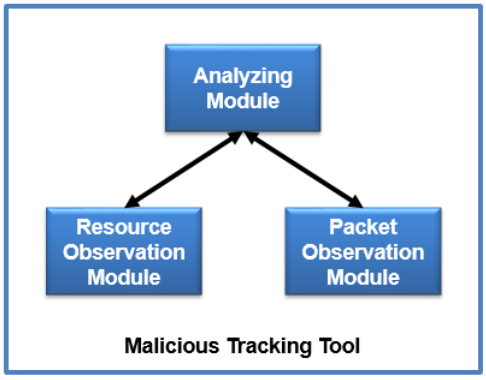


Figure 3: Modules in the malicious tracking tool.

3.3 Direct Method: Injection of Malicious Detecting Tool

As mentioned above, a middlebox is normally created or defined with a fresh operating system. If attackers were to modify the service function or inject malicious programs into middleboxes to perform attacks, this action could leave a trace. For example, if an abnormal process was installed on the middlebox, this process would consume a significant amount of CPU and memory because of harmful actions

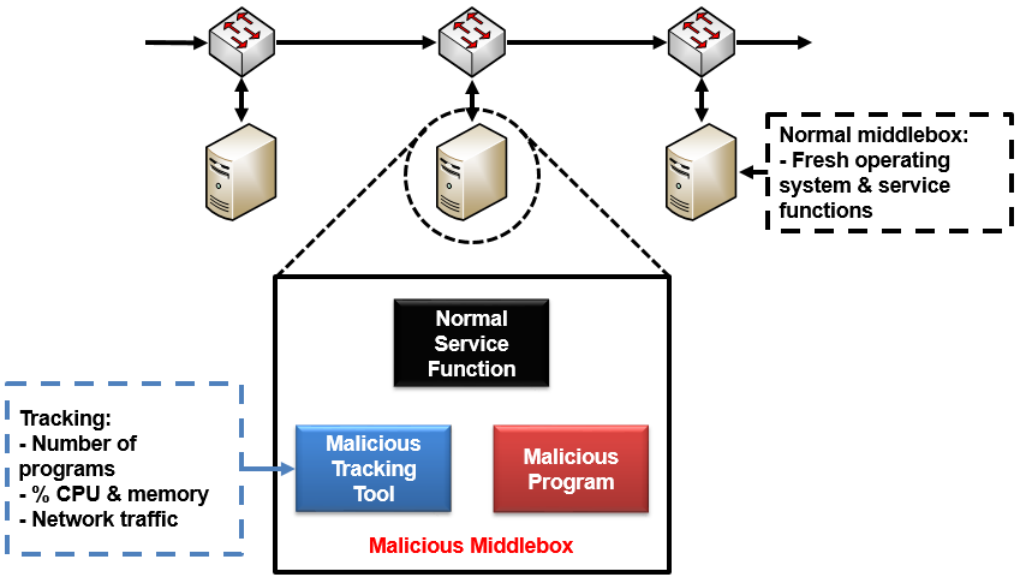


Figure 4: Direct method: injection of malicious detecting into middlebox.

(e.g., packet handling or packet forwarding), which would be easy to detect by observing the resource usage of the computer. On the other hand, if the service functions were to be modified to launch attacks, this action could also be detected by comparing the typical resource consumption between similar middleboxes. Other malicious actions could be discovered by observing the network traffic on network interfaces (e.g., packet dropping or multiplying attacks).

Our proposed method is designed to inject and run the malicious tracking tool on middleboxes to detect the above-mentioned types of attacks. To reduce the overhead for middleboxes, we can run this program either in a random period or run it consecutively to perform real-time detection. All of the detection results are sent to the controller. An example of the implementation of this method is depicted in Fig. 4.

3.4 Indirect Method: Probe Chain Generation

In the event of our malicious tracking tool being detected and even compromised to spoof the detecting results, our direct method and other proposed solutions would not be effective. We therefore decided to use another approach to solve this problem. We created two trustful middleboxes and connected them to another middlebox to form a probe service chain. We also installed a malicious tracking tool in the trustful middleboxes to observe the network traffic. This approach entails using the malicious tracking tool to analyze the network traffic to discover attacks. The trustful middleboxes are regenerated periodically to prevent the potential protection from being compromised, and the middlebox under testing is also chosen randomly.

This method is depicted in Fig. 5, in which case the probe chain is $Middlebox_1 - Middlebox_x - Middlebox_2$. For example, we send 100 packets from trustful $Middlebox_1$ to under testing $Middlebox_x$. These packets are intended to be processed by the service function inside $Middlebox_x$ and then be forwarded to next-hop trustful $Middlebox_2$. After a period, if $Middlebox_2$ receives only 90 packets (packet dropping attack), or 150 packets (packet multiplying attack), or receives a packet after a longer time than usual, this may be a man-in-the-middle or an eavesdropping attack. The detection results are also sent to the controller. Because we create trustful middleboxes to handle the detection mechanism, we have

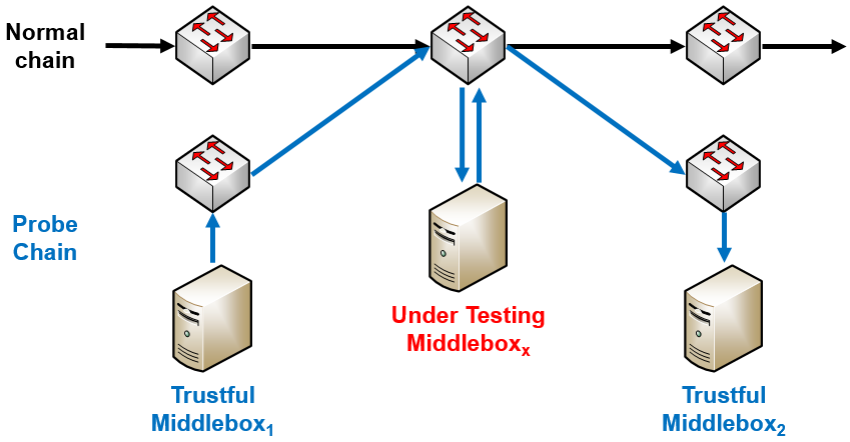


Figure 5: Indirect method: creation of a probe chain to test the operation of middleboxes

sufficient hardware resources and reduce the computational burden of the controller.

4 Implementation and Evaluation

In this section, we discuss the evaluation of the performance, efficiency, and applicability of our mechanisms by presenting the results of two experiments. In the first experiment, we evaluated the performance of our malicious tracking tool by calculating the CPU and memory consumption on the middleboxes. In the second experiment, we measured the detection rate, accuracy, and false alarm rate to evaluate the efficiency of our solution. Our mechanisms were implemented on Ubuntu 18.04 with 12 CPUs with Intel i7-8700 3.4GHz processors and 48 GB memory. OpenStack was installed with the Service Function Chaining feature to create the service chain. Each middlebox is a VM with one CPU and 2 GB memory. The malicious tracking tool is programmed in C with fewer than 500 lines of code.

4.1 Resources Consumption

Our methods involve the installation of the malicious tracking tool on a middlebox, which has limited computing resources. To evaluate the performance and applicability of our solutions, we calculated the CPU and memory consumption of our tool on middleboxes. When the direct method was implemented, our program consumed only 7.013% of the CPU and 0.0068% of the memory resources (2 GB in total) on average. These measurements are shown in Fig. 6 and 7 as a function of time. In the case of the indirect method, our program consumes even less resources than the direct method. These results confirm that our mechanism is effective with noticeable obstacles for middleboxes.

4.2 Efficiency

In this experiment, we generated a few of the attacks mentioned in section III-A to evaluate the efficiency of our mechanisms. The attack models are depicted in Fig. 8. We first created both trustful and normal middleboxes, connected them to form service chains and probe chains, and then modified one middlebox to play the role of a malicious one. For example, the malicious middlebox receives a% network traffic from $Switch_x$ but returns only b%. A packet dropping attack happens when $b < a$, and a packet multiplying attack occurs when $b > a$, which can easily be detected by observing the network traffic. If the

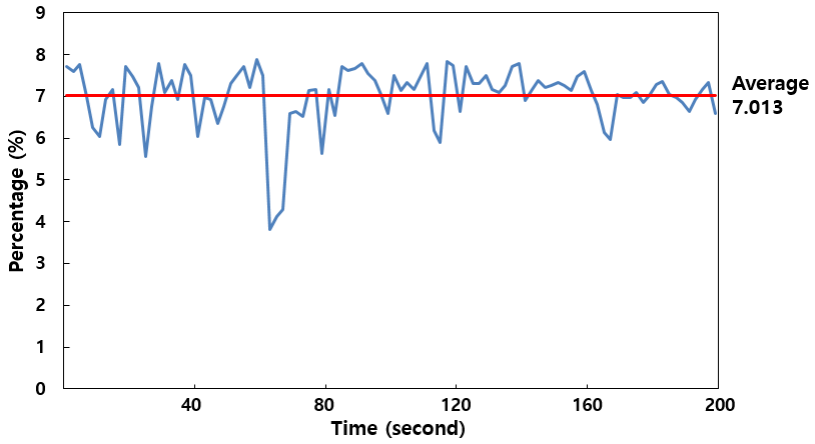


Figure 6: CPU usage of our tool on middlebox during the experiment.

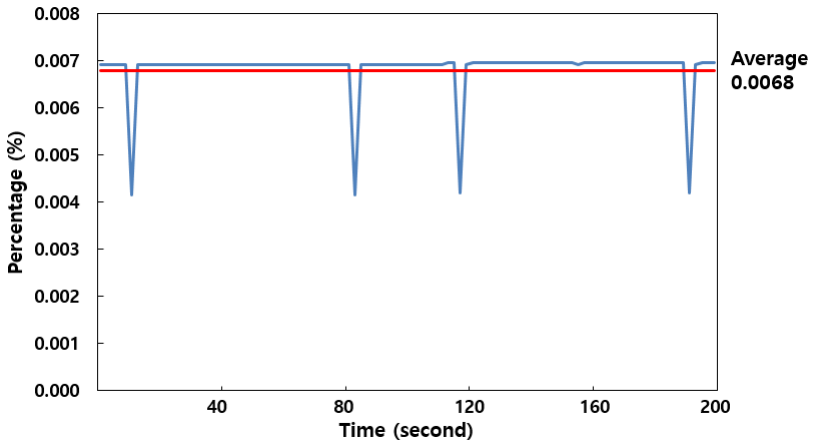


Figure 7: Memory usage of our tool on middlebox during the experiment.

malicious middlebox were to forward packets to attackers in an abnormal way (eavesdropping or man-in-the-middle attacks), these actions could be detected by calculating the transmission delay between the malicious and the trustful middleboxes. A resource abusing attack could also be detected by observing the resource consumption by the malicious middlebox itself.

The experimental results were categorized into four types: True Positive (TP) is the number of malicious middleboxes that are detected. False Positive (FP) is the number of normal middleboxes that are detected as malicious middleboxes. False Negative (FN) is the number of malicious middleboxes that are detected as normal middleboxes, and True Negative (TN) is the number of normal middleboxes that are detected as normal. The Detection Rate, Accuracy, and False Alarm Rate were used to evaluate the efficiency of our system, and were calculated to be 97.3958%, 99.6233%, and 0.3623% respectively. Compared with other solutions, our mechanism is able to effectively detect malicious middleboxes.

5 Conclusion and Future Work

A malicious middlebox is a substantial problem for Service Function Chaining. In this paper, we presented a scheme that can detect malicious middleboxes in SFC. Our proposed scheme combines two

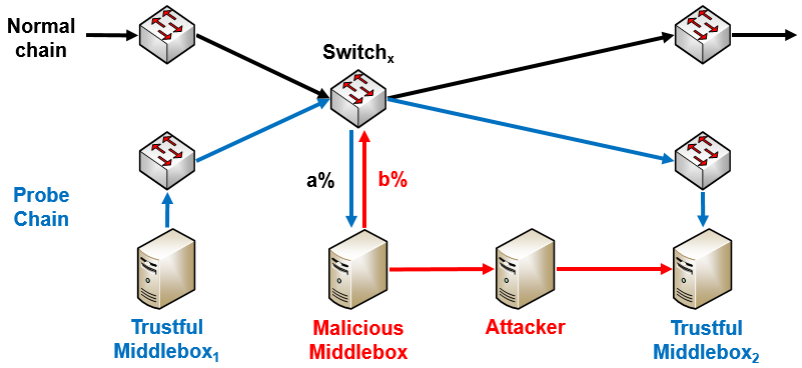


Figure 8: Attack models used in the experiments.

mechanisms: direct and indirect, to detect attacks that occur both internally and externally to middleboxes. The scheme therefore overcomes the disadvantages of other solutions and provides security for SFC. The experimental results validated the ability of our mechanism to detect malicious attacks with low resource consumption, a high detection rate, and high accuracy. In the future, we aim to improve our mechanism to increase its performance and efficiency, use machine learning to analyze the network data, and cover additional attack scenarios involving malicious middleboxes.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No 2018-0-00254, SDN security technology development).

References

[1] D. Bh, R. Jain, M. Samaka, and A. Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75(C):138–155, November 2016.

[2] M. Daghmechi Firoozjaei, J. Jeong, H. Ko, and H. Kim. Security challenges with network functions virtualization. *Future Generation Computer Systems*, 67:315–324, February 2017.

[3] L. Guo, J. Pang, and A. Walid. Dynamic service function chaining in sdn-enabled networks with middleboxes. In *Proc. of the 24th IEEE International Conference on Network Protocols (ICNP'16), Singapore, Singapore*, pages 1–10. IEEE, November 2016.

[4] A. Ibrahim, J. Hamlyn-Harris, J. Grundy, and M. Almorsy. Cloudsec: A security monitoring appliance for virtual machines in the iaas cloud model. In *Proc. of the 5th International Conference on Network and System Security (NSS'11), Milan, Italy*, pages 113–120. IEEE, October 2011.

[5] M. Jamshed, J. Lee, S. Moon, I. Yun, D. Kim, S. Lee, Y. Yi, and K. Park. Kargus: A highly-scalable software-based intrusion detection system. In *Proc. of the 2012 ACM Conference on Computer and Communications Security (CCS'12), New York, USA*, pages 317–328. ACM, October 2012.

[6] C.-T. Kuo, P.-W. Chi, V. Chang, and C.-L. Lei. Sfaas: Keeping an eye on iot fusion environment with security fusion as a service. *Future Generation Computer Systems*, 86:1424–1436, September 2018.

[7] Y. Li and M. Chen. Software-defined network function virtualization: A survey. *IEEE Access*, 3:2542–2553, December 2015.

[8] A. Medhat, T. Taleb, A. Elmangoush, G. Carella, S. Covaci, and T. Magedanz. Service function chaining in next generation networks: State of the art and research challenges. *IEEE Communications Magazine*, 55(2):216–223, October 2016.

[9] G. MIRJALILY and Z. LUO. Optimal network function virtualization and service function chaining: A survey. *Chinese Journal of Electronics*, 27(4):704–717, September 2018.

[10] C. Modi, D. Patel, B. Borisaniya, A. Patel, and M. Rajarajan. A novel framework for intrusion detection in cloud. In *Proc. of the 5th International Conference on Security of Information and Networks (SIN'12), New York, USA*, pages 67–74. ACM, October 2012.

[11] M. Pattaranantakul, R. He, Q. Song, Z. Zhang, and A. Meddahi. Nfv security survey: From use case driven threat analysis to state-of-the-art countermeasures. *IEEE Communications Surveys & Tutorials*, 20(4):3330–3368, July 2018.

[12] Z. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. Simple-fying middlebox policy enforcement using sdn. In *Proc. of the 2013 ACM SIGCOMM Computer Communication Review (SIGCOMM'13), New York, USA*, volume 43, pages 27–38. ACM, August 2013.

[13] P. Shanmugam, N. Subramanyam, J. Breen, C. Roach, and J. Merwe. Deidtect: towards distributed elastic intrusion detection. In *Proc. of the 2014 ACM SIGCOMM workshop on Distributed cloud computing (DCC'14), New York, USA*, pages 17–24. ACM, August 2014.

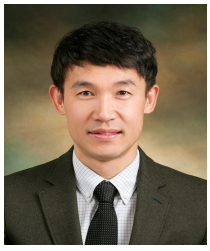
[14] J. Sherry, C. Lan, R. Popa, and S. Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *Proc. of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'15), New York, USA*, pages 213–226. ACM, August 2015.

[15] M. Watson, S. N. Shirazi, A. Marnerides, A. Mauthe, and D. Hutchison. Malware detection in cloud computing infrastructures. *IEEE Transactions on Dependable and Secure Computing*, 13(2):192–205, July 2015.

[16] W. Yang and C. Fung. A survey on security in network functions virtualization. In *Proc. of the 2016 IEEE NetSoft Conference and Workshops (NetSoft'16), Seoul, South Korea*, pages 15–19. IEEE, June 2016.

Author Biography

Nguyen Canh Thang received the B.S. degree in control engineering and automation from the Hanoi University of Science and Technology, Hanoi, Vietnam, in 2018. He is currently pursuing a Master's degree in information and communication at Soongsil University, Seoul, South Korea. His research interests include software-defined networks, service function chaining, and network security.



Minho Park received the B.S. and M.S. degrees in electronics engineering from Korea University in 2000 and 2002, respectively, and the Ph.D. degree from the School of Electrical Engineering and Computer Science, Seoul National University, Seoul, South Korea, in 2010. During this time, he spent two years with Samsung Electronics from 2002 to 2004, where he eventually held the position of Senior Engineer with the 3GPP LTE S/W Development Group. Starting in 2011, he completed two years of postdoctoral research at Carnegie Mellon University (CMU). He is currently an Assistant Professor with the School of Electronic Engineering, Soongsil University, Seoul. His current research interests include wireless networks, vehicular communication networks, network security, and cloud computing.